Recommendation_system Predict

Masilo Ramatseba

In [3]:
```
# Import several modules and packges
import numpy as np
import pandas as pd
import scipy as sp
import matplotlib.pyplot as plt
import seaborn as sns


from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer

import operator
import heapq

from surprise import Reader, Dataset
from surprise.model_selection import cross_validate, train_test_split, GridSearchCV

import warnings
warnings.filterwarnings('ignore')
```

Loading of the data

In [6]:
```
train_df=pd.read_csv('train.csv')
test_df=pd.read_csv('test.csv')
links_df=pd.read_csv('links.csv')
imdb_df=pd.read_csv('imdb_data.csv')
gtags_df=pd.read_csv('genome_tags.csv')
gscores_df=pd.read_csv('genome_scores.csv')
movies_df=pd.read_csv('movies.csv')
tags_df = pd.read_csv('tags.csv')
sample_submission=pd.read_csv('sample_submission.csv')
```

In [ ]:
```
train_df=train_df.drop('timestamp',axis=1)
train_df
```

Data processing

In [7]:
```
#Taking a look at our data frames

display("movies", movies_df.head())
display("imdb", imdb_df.head())
display("train", train_df.head())
display("test", test_df.head())
display("genome scores", gscores_df.head())
display("genome tags", gtags_df.head())
display("tags", tags_df.head())
display("links", links_df.head())
```

'movies'

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

'imdb'

	movieId	title_cast	director	runtime	budget	plot_keywords
0	1	Tom Hanks|Tim Allen|Don Rickles|Jim Varney|Wal...	John Lasseter	81.0	$30,000,000	toy|rivalry|cowboy|cgi animation
1	2	Robin Williams|Jonathan Hyde|Kirsten Dunst|Bra...	Jonathan Hensleigh	104.0	$65,000,000	board game|adventurer|fight|game
2	3	Walter Matthau|Jack Lemmon|Sophia Loren|Ann-Ma...	Mark Steven Johnson	101.0	$25,000,000	boat|lake|neighbor|rivalry
3	4	Whitney Houston|Angela Bassett|Loretta Devine|...	Terry McMillan	124.0	$16,000,000	black american|husband wife relationship|betra...
4	5	Steve Martin|Diane Keaton|Martin Short|Kimberl...	Albert Hackett	106.0	$30,000,000	fatherhood|doberman|dog|mansion

'train'

	userId	movieId	rating	timestamp
0	5163	57669	4.0	1518349992
1	106343	5	4.5	1206238739
2	146790	5459	5.0	1076215539
3	106362	32296	2.0	1423042565
4	9041	366	3.0	833375837

'test'

	userId	movieId
0	1	2011
1	1	4144
2	1	5767
3	1	6711
4	1	7318

'genome scores'

	movieId	tagId	relevance
0	1	1	0.02875
1	1	2	0.02375
2	1	3	0.06250
3	1	4	0.07575
4	1	5	0.14075

'genome tags'

	tagId	tag
0	1	007
1	2	007 (series)
2	3	18th century
3	4	1920s
4	5	1930s

'tags'

	userId	movieId	tag	timestamp
0	3	260	classic	1439472355
1	3	260	sci-fi	1439472256
2	4	1732	dark comedy	1573943598
3	4	1732	great dialogue	1573943604
4	4	7569	so bad it's good	1573943455

'links'

	movieId	imdbId	tmdbId
0	1	114709	862.0
1	2	113497	8844.0
2	3	113228	15602.0
3	4	114885	31357.0
4	5	113041	11862.0

Shapes of the Dataframes

In [105…

```
# Now we take a look at the shapes of the DataFrames
shape = {'Genome scores': gscores_df.shape,
            'Genome tags':gtags_df.shape,
            'IMDB': imdb_df.shape,
            'Links': links_df.shape,
            "Movies": movies_df.shape,
            "tags": tags_df.shape,
            'Train': train_df.shape,
            'Test': test_df.shape}

df_shape = pd.DataFrame(list(shape.items()), columns=['Dataframe', 'Shape'])
df_shape
```

Out[105…

	Dataframe	Shape
0	Genome scores	(15584448, 3)
1	Genome tags	(1128, 2)
2	IMDB	(27278, 6)
3	Links	(62423, 3)
4	Movies	(62423, 3)
5	tags	(1093360, 3)
6	Train	(10000038, 3)
7	Test	(5000019, 3)

Check for null values

In [174…

```
# Checking for null values in the DataFrames

display("movies", movies_df.isnull().sum())
print('====================================')
display("imdb", imdb_df.isnull().sum())
print('====================================')
display("train", train_df.isnull().sum())
print('====================================')
display("test", test_df.isnull().sum())
print('====================================')
display("genome scores", gscores_df.isnull().sum())
print('====================================')
display("genome tags", gtags_df.isnull().sum())
print('====================================')
display("links", links_df.isnull().sum())
```

```
'movies'
movieId    0
title      0
genres     0
dtype: int64
====================================
'imdb'
movieId             0
title_cast      10068
director         9874
runtime         12089
budget          19372
plot_keywords   11078
dtype: int64
====================================
'train'
userId     0
movieId    0
rating     0
dtype: int64
====================================
'test'
```

```
userId     0
movieId    0
Id         0
dtype: int64
====================================
'genome scores'
movieId      0
tagId        0
relevance    0
dtype: int64
====================================
'genome tags'
tagId    0
tag      0
dtype: int64
====================================
'links'
movieId      0
imdbId       0
tmdbId     107
dtype: int64
```

Minimum and Maximum ratings recieved

In [10]:
```
display("lowest rating", train_df.rating.min())
display("highest rating", train_df.rating.max())
```

```
'lowest rating'
0.5
'highest rating'
5.0
```

Getting our data ready for Exploratory Data Analysis(EDA)

In [11]:
```
# Merging datasets for EDA

# Creating a dataframe where we select the features which would be important for our analysis
df_merge = imdb_df[['movieId','title_cast','director', 'plot_keywords']]
df_merge = df_merge.merge(movies_df[['movieId', 'genres', 'title']], on='movieId', how='inner')

#Add columnn for release year
df_merge['year'] = df_merge['title'].str.extract(r"\((\d+)\)", expand=False)
df_merge.head()
```

Out[11]:

	movieId	title_cast	director	plot_keywords	genres	title	year
0	1	Tom Hanks|Tim Allen|Don Rickles|Jim Varney|Wal...	John Lasseter	toy|rivalry|cowboy|cgi animation	Adventure|Animation|Children|Comedy|Fantasy	Toy Story (1995)	1995
1	2	Robin Williams|Jonathan Hyde|Kirsten Dunst|Bra...	Jonathan Hensleigh	board game|adventurer|fight|game	Adventure|Children|Fantasy	Jumanji (1995)	1995
2	3	Walter Matthau|Jack Lemmon|Sophia Loren|Ann-Ma...	Mark Steven Johnson	boat|lake|neighbor|rivalry	Comedy|Romance	Grumpier Old Men (1995)	1995
3	4	Whitney Houston|Angela Bassett|Loretta Devine|...	Terry McMillan	black american|husband wife relationship|betra...	Comedy|Drama|Romance	Waiting to Exhale (1995)	1995
4	5	Steve Martin|Diane Keaton|Martin Short|Kimberl...	Albert Hackett	fatherhood|doberman|dog|mansion	Comedy	Father of the Bride Part II (1995)	1995

Prepare selected features for EDA

In [12]:
```
# Convert data types to strings for string handling
df_merge['title_cast'] = df_merge.title_cast.astype(str)
df_merge['plot_keywords'] = df_merge.plot_keywords.astype(str)
df_merge['genres'] = df_merge.genres.astype(str)
df_merge['director'] = df_merge.director.astype(str)
```

In [13]:
```
# Removing spaces and converting to lowercase
```

```
df_merge['director'] = df_merge['director'].apply(lambda x: "".join(x.lower() for x in x.split(' ')))
df_merge['plot_keywords'] = df_merge['plot_keywords'].apply(lambda x: "".join(x.lower() for x in x.split(' ')))
```

In [15]:
```
# Discarding the pipes between the plot keywords' and extracting only the first five words

df_merge['plot_keywords'] = df_merge['plot_keywords'].map(lambda x: x.split('|')[:5])
df_merge['plot_keywords'] = df_merge['plot_keywords'].apply(lambda x: " ".join(x))
```

In [16]:
```
# Discarding the pipes between the genres and convert to lowercase

df_merge['genres'] = df_merge['genres'].map(lambda x: x.lower().split('|'))
df_merge['genres'] = df_merge['genres'].apply(lambda x: " ".join(x))
```

In [17]:
```
# removing punctuation from title cast
import string


def remove_punctuation(message):
    return ''.join([l.lower() for l in message if l not in string.punctuation])


df_merge['title_cast'] = df_merge['title_cast'].apply(remove_punctuation)
```

In [18]:
```
df_merge.head()
```

Out[18]:

	movieId	title_cast	director	plot_keywords	genres	title	year
0	1	tom hankstim allendon ricklesjim varneywallace...	johnlasseter	toy rivalry cowboy cgianimation	adventure animation children comedy fantasy	Toy Story (1995)	1995
1	2	robin williamsjonathan hydekirsten dunstbradle...	jonathanhensleigh	boardgame adventurer fight game	adventure children fantasy	Jumanji (1995)	1995
2	3	walter matthaujack lemmonsophia lorenannmargre...	markstevenjohnson	boat lake neighbor rivalry	comedy romance	Grumpier Old Men (1995)	1995
3	4	whitney houstonangela bassettloretta devinelel...	terrymcmillan	blackamerican husbandwiferelationship betrayal...	comedy drama romance	Waiting to Exhale (1995)	1995
4	5	steve martindiane keatonmartin shortkimberly w...	alberthackett	fatherhood doberman dog mansion	comedy	Father of the Bride Part II (1995)	1995

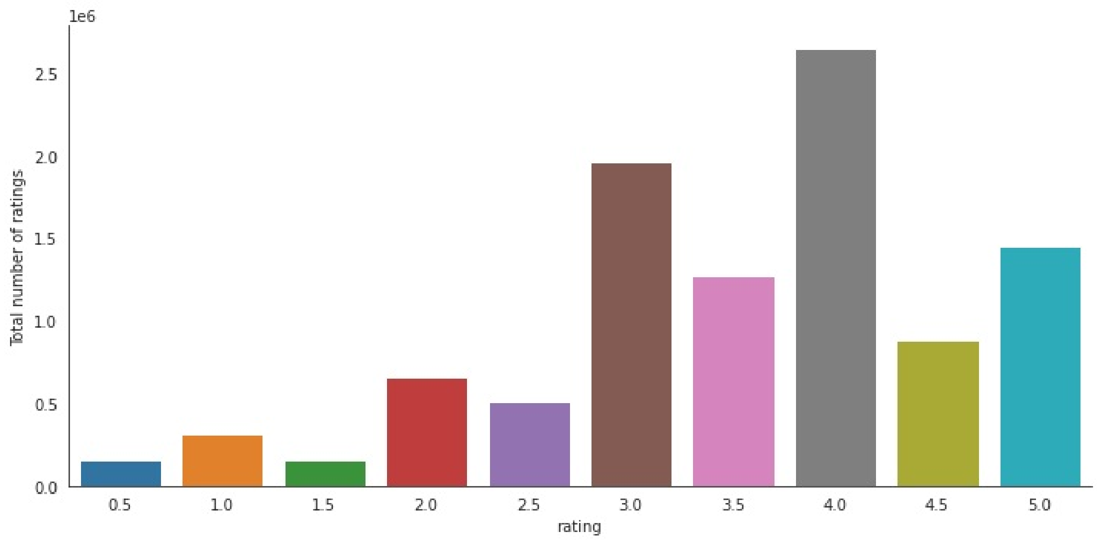
Exploratory Data Analysis(EDA)

In [20]:
```
year = df_merge['year']
movieId = df_merge['movieId']
```

In [21]:
```
# creating a graph representing the total number of ratings for each category

with sns.axes_style('white'):
    g = sns.catplot("rating", data=train_df, aspect=2.0,kind='count')
    g.set_ylabels("Total number of ratings")
print (f'Average rating in dataset: {np.mean(train_df["rating"])}')
```

```
Average rating in dataset: 3.5333951730983424
```



Observation and Interpretation:

-The average rating is 3.5

-The most movies are rated 4.0, followed by 3.0

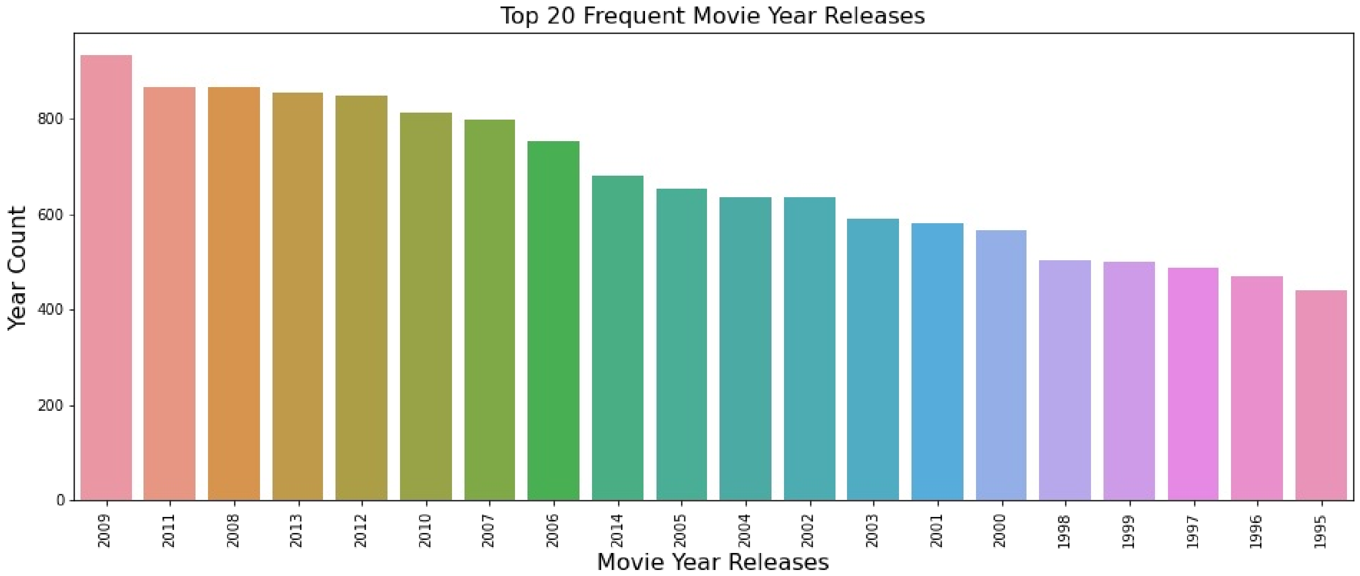
-The range is from 0.5 to 5.0 as a rating

In [23]:
```
# creating a graph indicating the top 20 years with the most movie releases

plt.figure(figsize=(16,6))
sns.countplot(x='year', data=df_merge,order=df_merge['year'].value_counts().iloc[:20].index)
plt.title('Top 20 Frequent Movie Year Releases', fontsize=16)
plt.xticks(rotation=90)

#add axis labels
plt.xlabel('Movie Year Releases',fontsize=16)
plt.ylabel('Year Count',fontsize=16)
```

Out[23]:
```
Text(0, 0.5, 'Year Count')
```



Observation and Interpretation:

- The year with the highest count of released movies was 2009

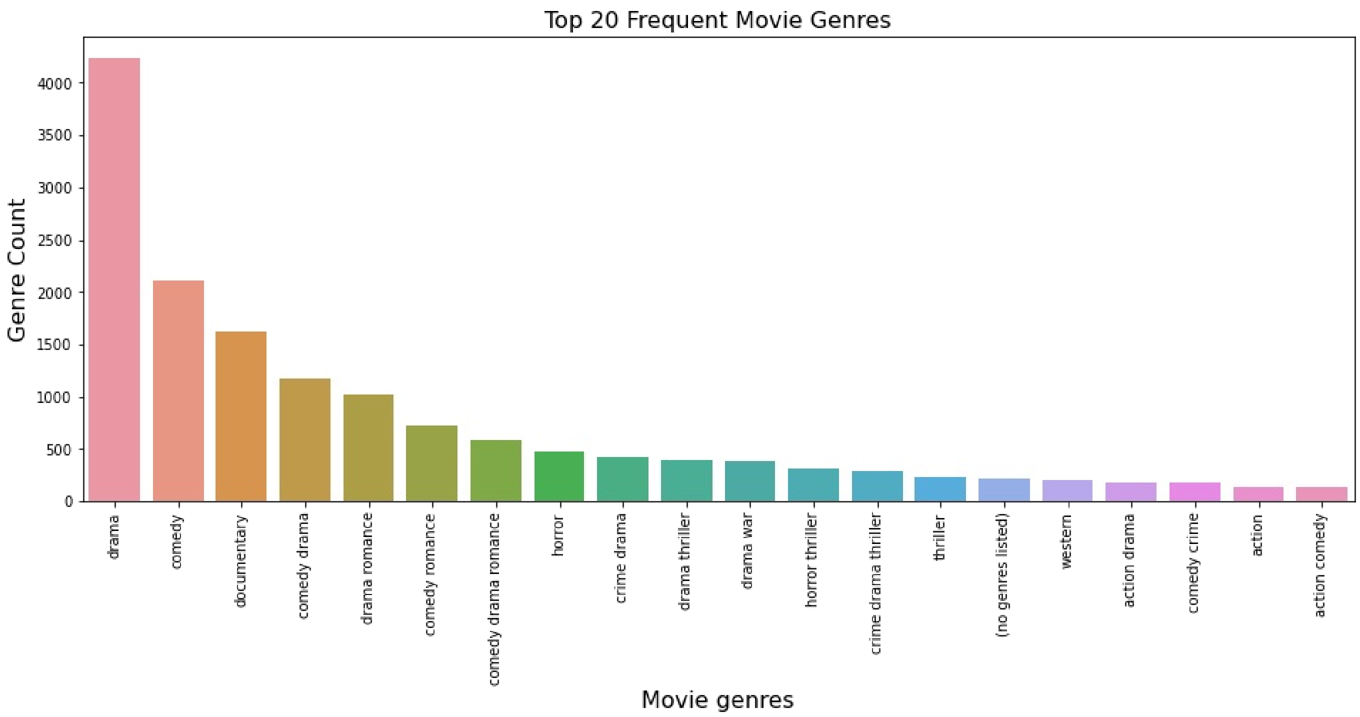
- The year with the lowest count of released movies was 1995

In [25]:
```
# creating a graph indicating the top 20 most frequently occuring movie genres

plt.figure(figsize=(16,6))
sns.countplot(x='genres', data=df_merge,order=df_merge['genres'].value_counts().iloc[:20].index)
plt.title('Top 20 Frequent Movie Genres', fontsize=16)
plt.xticks(rotation=90)

#add axis labels
plt.xlabel('Movie genres',fontsize=16)
plt.ylabel('Genre Count',fontsize=16)
```

Out[25]:
```
Text(0, 0.5, 'Genre Count')
```



What can we observe from the graph?

-We can see the the genre with the most movies apears to be drama followed by comedy

-Action comedy seems to be less frequent

Feature engineering

In [27]:
```
# Droping timestamps from the train and tags DataFrames
train_df = train_df.drop(['timestamp'], axis=1)
tags_df = tags_df.drop(['timestamp'],axis=1)
```

In [28]:
```
#Removing the dates from title so that we are only left with the title of the movie
df_merge['title'] = df_merge['title'].replace(to_replace=r'\(\d+\)', value='', regex=True)
df_merge['title'] = df_merge['title'].apply(lambda x: x.rstrip())
df_merge.head(3)
```

Out[28]:

	movieId	title_cast	director	plot_keywords	genres	title	year
0	1	tom hankstim allendon ricklesjim varneywallace...	johnlasseter	toy rivalry cowboy cgianimation	adventure animation children comedy fantasy	Toy Story	1995
1	2	robin williamsjonathan hydekirsten dunstbradle...	jonathanhensleigh	boardgame adventurer fight game	adventure children fantasy	Jumanji	1995
2	3	walter matthaujack lemmonsophia lorenannmargre...	markstevenjohnson	boat lake neighbor rivalry	comedy romance	Grumpier Old Men	1995

In [29]:
```
#TfidfVectorizer to convert text data into numerical data.
tfd_vect = TfidfVectorizer()
```

In [30]:
```python
#Convert plot_keywords by fitting it into the TfidVectorizer
vect_plot = tfd_vect.fit_transform(df_merge['plot_keywords'])
```

In [31]:
```python
# Fit TF-IDF into cosine similarity to take advantage of the strengths of both methods and compare the similari
cosine_sim = cosine_similarity(vect_plot)
```

RECOMMEND MOVIES BASED ON A GIVEN MOVIE

In [32]:
```python
def recommend_movies(title):
    # Find the index of the movie that matches the title
    indx = df_merge[df_merge['title'] == title].index[0]

    # Get the pairwise similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[indx]))

    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:11]

    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar movies
    return df_merge['title'].iloc[movie_indices]
```

In [34]:
```python
print(recommend_movies("Jumanji"))
```

```
3384                                Road to El Dorado, The
9397                                             Word Wars
18512              Under the Boardwalk: The Monopoly Story
10398                                              Zathura
1566                                             Game, The
15635                                                 Glue
9281     Springtime in a Small Town (Xiao cheng zhi chun)
14789                                       Wild Hunt, The
23045                                           Another Me
24798                                            Forgotten
Name: title, dtype: object
```

RECOMMEND MOVIES BASED ON A GIVEN YEAR

In [35]:
```python
def recommend_year(year):
    # Find the index of the movie that matches the title
    idx = df_merge[df_merge['year'] == year].index[0]

    # Get the pairwise similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:11]

    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar movies
    return df_merge['title'].iloc[movie_indices]
# Example usage: this will recommend movies around t
```

In [37]:
```python
# Example usage: this will recommend movies around the same period of the movie previously watched

print(recommend_year('2014'))
```

```
2679           Detroit Rock City
16857                       Prom
23063                  Free Ride
493                    Mr. Jones
14203                    Hum Tum
15072                     Tekken
834                      Tin Cup
7125     Where the Day Takes You
24303              Attila Marcel
20778        Temptation (Tentação)
Name: title, dtype: object
```

Collaborative based filtering

In [38]:
```
# merge df_train and df_merge datasets
df_collab = pd.merge(train_df, df_merge, on='movieId', how='left')
```

In [39]:
```
# drop redundant featuers in our new df_collab dataset
df_collab = df_collab.drop(['title_cast','director','plot_keywords','genres','year'],axis=1)
```

In [137…
```
#work with the first 500 thousand rows as the original dataset is too large and causes computational complexity
df_collab = df_collab.iloc[0:10000]
```

In [138…
```
#display the new dataset
df_collab.head()
```

Out[138…

	userId	movieId	rating	title
0	5163	57669	4.0	In Bruges
1	106343	5	4.5	Father of the Bride Part II
2	146790	5459	5.0	Men in Black II (a.k.a. MIIB) (a.k.a. MIB 2)
3	106362	32296	2.0	Miss Congeniality 2: Armed and Fabulous
4	9041	366	3.0	Wes Craven's New Nightmare (Nightmare on Elm S...

In [139…
```
# pivot table to group data by one or more variables and to summarize the data by calculating various aggregate
# and average.
utility_matrix = df_collab.pivot_table(index=['userId'],
                                       columns=['title'],
                                       values='rating')
utility_matrix.shape
```

Out[139… (8735, 3362)

In [140…
```
utility_matrix.head(3)
```

Out[140…

title	Days of Summer	*batteries not included	...And Justice for All	10 Things I Hate About You	10,000 BC	101 Dalmatians	101 Dalmatians (One Hundred and One Dalmatians)	102 Dalmatians	12 Angry Men	12 Years a Slave	...	Zodiac	Zombeavers
userId													
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
12	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
59	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN

3 rows × 3362 columns

In [141…
```
import scipy as sp

# Normalize each row (a given user's ratings) of the utility matrix

util_matrix_norm = utility_matrix.apply(lambda x: (x-np.mean(x))/(np.max(x)-np.min(x)), axis=1)

# Fill Nan values with 0's, transpose matrix, and drop users with no ratings

util_matrix_norm.fillna(0, inplace=True)
util_matrix_norm = util_matrix_norm.T
util_matrix_norm = util_matrix_norm.loc[:, (util_matrix_norm != 0).any(axis=0)]

# Save the utility matrix in scipy's sparse matrix format
util_matrix_sparse = sp.sparse.csr_matrix(util_matrix_norm.values)
```

In [142…
```
# Compute the similarity matrix using the cosine similarity metric

user_similarity = cosine_similarity(util_matrix_sparse.T)

# Save the matrix as a dataframe to allow for easier indexing

user_sim_df = pd.DataFrame(user_similarity,
                           index = util_matrix_norm.columns,
                           columns = util_matrix_norm.columns)
```

In [162…
```
user_sim_df.head(3)
```

Out[162...

userId	606	642	1123	1977	2316	3092	3144	3367	3394	3503	...	160922	161113	161115	161472	161660	161919	162047
userId																		
606	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
642	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1123	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0

3 rows × 603 columns

In [143...

```
def collab_recommendations(user, N=10, k=20):
    # Cold-start problem - no ratings given by the reference user.
    # With no further user data, we solve this by simply recommending
    # the top-N most popular books in the item catalog.
    if user not in user_sim_df.columns:
        return df_collab.groupby('title').mean().sort_values(by='rating',
                                        ascending=False).index[:N].to_list()

    # Gather the k users which are most similar to the reference user
    sim_users = user_sim_df.sort_values(by=user, ascending=False).index[1:k+1]
    favorite_user_items = [] # <-- List of highest rated items gathered from the k users
    most_common_favorites = {} # <-- Dictionary of highest rated items in common for the k users

    for i in sim_users:
        # Maximum rating given by the current user to an item
        max_score = util_matrix_norm.loc[:, i].max()
        # Save the names of items maximally rated by the current user
        favorite_user_items.append(util_matrix_norm[util_matrix_norm.loc[:, i]==max_score].index.tolist())

    # Loop over each user's favorite items and tally which ones are
    # most popular overall.
    for item_collection in range(len(favorite_user_items)):
        for item in favorite_user_items[item_collection]:
            if item in most_common_favorites:
                most_common_favorites[item] += 1
            else:
                most_common_favorites[item] = 1
    # Sort the overall most popular items and return the top-N instances
    sorted_list = sorted(most_common_favorites.items(), key=operator.itemgetter(1), reverse=True)[:N]
    top_N = [x[0] for x in sorted_list]
    return top_N
```

In [144...

```
collab_recommendations(314)
```

Out[144...

```
['Kokowääh',
 'Diary of a Wimpy Kid: Rodrick Rules',
 'All Is Lost',
 'Insider, The',
 'Menace II Society',
 'Umberto D.',
 "Ulee's Gold",
 'Fearless',
 'Men of Honor',
 'Stage Beauty']
```

Model creation

In [164...

```
# The Reader class is used to convert the input data into a Dataset object, which can then be used to train the
reader = Reader(rating_scale=(1, 5))
```

In [165...

```
# code is used to load a pandas DataFrame into a Dataset object in the Surprise library that can be used to tra
data = Dataset.load_from_df(train_df[['userId', 'movieId', 'rating']], reader)
```

In [166...

```
# splitting the dataset into train and test

trainset, testset = train_test_split(data, test_size=0.2)#, random_state=42)
```

Singular value decomposition(SVD) model

-SVD is a powerful tool for dimensionality reduction, feature extraction, and data compression in machine learning.

In [167...

```
# importing the model
from surprise import SVD
```

In [168...

```
# creating the model object
svd_model = SVD(n_epochs = 40, n_factors = 400, init_std_dev = 0.005, random_state=42)
```

```
# fitting the data
svd_model.fit(data.build_full_trainset())

# making predictions
predictions_svd = svd_model.test(testset)
```

Non-negative Matrix Factorization (NMF)

- Non-Negative Matrix Factorization is useful when there are many attributes and the attributes are ambiguous or have weak predictability.

In [127…
```
# importing the model
from surprise import NMF

#creating the model object
nmf_model = NMF()

# Train the model
nmf_model.fit(trainset)

# Make predictions on the test set
predictions_nmf = nmf_model.test(testset)
```

CO-CLUSTERING

CoClustering: This algorithm is based on co-clustering of rows and columns in the ratings matrix. It is used for both item-based and user-based collaborative filtering.

In [128…
```
# importing the model
from surprise import CoClustering

# creating the model object
cocl_model = CoClustering()

# Train the model
cocl_model.fit(trainset)

# Make predictions on the test set
predictions_co = cocl_model.test(testset)
```

Check for model perfomance

- Now that we have created our models we'll evaluate thier performance

In [129…
```
#import accuracy from surprise to measure accuracy
from surprise import accuracy
```

In [130…
```
# calculating the accuracy of each model based on the RMSE score

accuracy.rmse(predictions_svd)
accuracy.rmse(predictions_nmf)
accuracy.rmse(predictions_co)
```

```
RMSE: 0.6639
RMSE: 0.8875
RMSE: 0.9010
```

Out[130…
```
0.9009714658445606
```

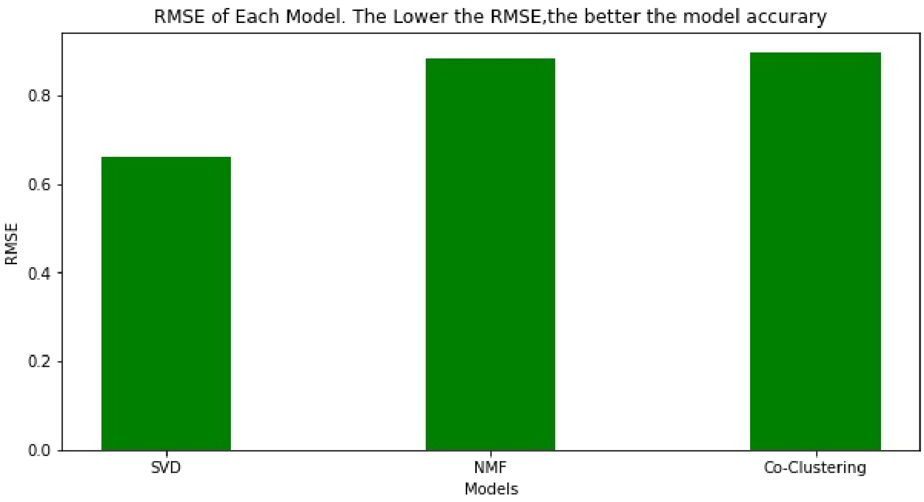
Now we'll visualise the RMSE's of the models

In [115…
```
data = {'SVD':0.6630, 'NMF':0.8842, 'Co-Clustering':0.8973}
models_md = list(data.keys())
values_md = list(data.values())

fig = plt.figure(figsize = (10, 5))

# creating the bar plot
plt.bar(models_md, values_md, color ='green',
        width = 0.4)

plt.xlabel("Models")
plt.ylabel("RMSE")
plt.title("RMSE of Each Model. The Lower the RMSE,the better the model accurary")
plt.show()
```



Interpretation of the models

I used three models for the recomendation system, which are the Singular Value Decomposition(SVD) , Non-negative Matrix Factorization(NMF) and Co-Clustering.

-The Singular value Decomposition technique is a method from linear algebra that has been generally used as a dimensionality reduction technique in machine learning,it is a matrix factorisation technique, which reduces the number of features of a dataset by reducing the space dimension from N-dimension to K-dimension (where K<N)

-Non-negative Matrix Factorization(NMF) is a method used to factorize a non-negative matrix, X, into the product of two lower rank matrices, A and B, such that AB approximates an optimal solution of X.

-Co-Clustering is based on co-clThese are several models that can handle large datasets with over 10 million rows for the movie recommendation systems.

-The SVD model had a lower RSME score than the NMF model and Co Clustering. The scores showed the optimal model to use which was the SVD Modelustering of rows and columns in the ratings matrix. It is used for both item-based and user-based collaborative filtering.

Conclusion

So according to our model evaluations, we can see that the SVD model out performs the other models by having the least RMSE which means it will get much accurate results. With that been said it can be concluded that the SVD is the chosen model for our recommendation system as it will give better predictions as compared to the other models.

Kaggle submission

In [170…
```
test_df['Id'] = test_df['userId'].astype(str) + "_" + test_df['movieId'].astype(str)
```

In [171…
```
predictions = []
for index, row in test_df.iterrows():
    pred = svd_model.predict(row['userId'], row['movieId'])
    predictions.append([row['Id'], pred.est])
```

In [172…
```
submission = pd.DataFrame(predictions, columns=["Id", "rating"])
submission.head()
```

Out[172…

	Id	rating
0	1_2011	3.063302
1	1_4144	4.186689
2	1_5767	3.769391
3	1_6711	4.125384
4	1_7318	2.980803

In [173…
```
submission.to_csv("submission_6.csv", index=False)
```

In [121…
```
submission.head()
```

Out[121…

	Id	rating
0	1_2011	3.288882
1	1_4144	4.219446
2	1_5767	3.885288
3	1_6711	4.243513
4	1_7318	3.388648